I began analysis by determining the shortest possible plan lengths per problem. Knowing that breadth-first search will always provide a solution with minimal plan length, I ran that first (solutions included):

```
Solving Air Cargo Problem 1 using breadth_first_search...

Expansions   Goal Tests   New Nodes
    43          56         180

Plan length: 6  Time elapsed in seconds: 0.020223473000442027
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

```
Solving Air Cargo Problem 2 using breadth_first_search...

Expansions   Goal Tests   New Nodes
   3401        4672       31049

Plan length: 9  Time elapsed in seconds: 8.939790221999829
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

```
Solving Air Cargo Problem 3 using breadth_first_search...

Expansions   Goal Tests   New Nodes
  14491       17947       128184

Plan length: 12  Time elapsed in seconds: 64.3768693510001
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)
```

I can see however that it takes quite a long time time to arrive at the solution for problem 3 with BFS. I next took a look at depth-first search thinking it might provide solutions more quickly, however with longer plans (solutions not included as they are quite long):

```
Solving Air Cargo Problem 1 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
    21          22          84

Plan length: 20  Time elapsed in seconds: 0.009400668999660411
```

```
Solving Air Cargo Problem 2 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
   1192        1193       10606

Plan length: 1138  Time elapsed in seconds: 5.623136508000243
```

```
Solving Air Cargo Problem 3 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
   2099        2100       17558

Plan length: 2014  Time elapsed in seconds: 14.396704421000322
```

DFS did indeed provide answers more quickly, however the plan lengths are massive! I next ran uniform cost search to see if it could provide an optimal solution more quickly than BFS:

```
Solving Air Cargo Problem 1 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
    55          57         224

Plan length: 6  Time elapsed in seconds: 0.024680762999196304
```

```
Solving Air Cargo Problem 2 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
   4761        4763       43206

Plan length: 9  Time elapsed in seconds: 7.215535057000125
```

```
Solving Air Cargo Problem 3 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
  17783       17785       155920

Plan length: 12  Time elapsed in seconds: 31.618546196999887
```

Overall, uniform cost search performed the best out of the 3 uninformed search algorithms I attempted- it found optimal solutions pretty quickly for all 3 problems:

Search Algorithm	Problem Number	Nodes Expanded	Goal Tests	Time (rounded)	Plan Length	Optimal Plan
BFS	1	43	56	0.02	6	✓
DFS	1	21	22	0.01	20	✗
Uniform Cost	1	55	57	0.02	6	✓
BFS	2	3,401	4,672	8.94	9	✓
DFS	2	1,192	1,193	5.62	1,138	✗
Uniform Cost	2	4,761	4,763	7.22	9	✓
BFS	3	14,491	17,947	64.38	12	✓
DFS	3	2,099	2,100	14.4	2,014	✗
Uniform Cost	3	17,783	17,785	31.62	12	✓

Next, I ran A* with the ignore ignore-preconditions heuristic:

```
Solving Air Cargo Problem 1 using astar_search with h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
    41          43         170

Plan length: 6  Time elapsed in seconds: 0.02474069299933035
```

```
Solving Air Cargo Problem 2 using astar_search with h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
   1450        1452       13303

Plan length: 9  Time elapsed in seconds: 2.734135979000712
```

```
Solving Air Cargo Problem 3 using astar_search with h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
   5003        5005       44586

Plan length: 12  Time elapsed in seconds: 10.776118698000573
```

And then A* with the level-sum heuristic:

```
Solving Air Cargo Problem 1 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
    11          13          50

Plan length: 6  Time elapsed in seconds: 0.594194327999503
```

```
Solving Air Cargo Problem 2 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
    86          88         841

Plan length: 9  Time elapsed in seconds: 124.8168282530005
```

```
Solving Air Cargo Problem 3 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
   311         313         2863

Plan length: 12  Time elapsed in seconds: 695.7802314920009
```

We can see the results for the different heuristics here:

A* Heuristic	Problem Number	Nodes Expanded	Goal Tests	Time (rounded)	Plan Length	Optimal Plan
ignore-preconditions	1	41	43	0.02	6	✓
level-sum	1	11	13	0.59	6	✓
ignore-preconditions	2	1,450	1,452	2.73	9	✓
level-sum	2	86	88	124.82	9	✓
ignore-preconditions	3	5,003	5,005	10.78	12	✓
level-sum	3	311	313	695.78	12	✓

Overall, levelsum was quite slow to perform due to the heavy cost of the calculation for the hueristic- but it was able to minimize the number of nodes that had to be examined suggesting that the heuristic was very good. Preconditions was a good balance in that the cost of calculating the heuristic was quite low and it did manage to lead the search generally in a good direction, allowing it to perform even faster than DFS while providing an optimal solution. Ignoring preconditions is a frequently used heuristic in planning search (Artificial Intelligence A Modern Approach, 3rd ed., Russell and Norvig, p. 376) as is level sum (Artificial Intelligence A

Modern Approach, 3rd ed., Russell and Norvig, p. 382), however we know from the prior class project that it can be better to pick a cheaper to calculate heuristic over a smarter but harder to calculate one (Artificial Intelligence Nanodegree, Build a Game-Playing Agent).